View metadata, citation and similar papers at core.ac.uk
brought to you by CORE
provided by University of Queensland eSpace

FPGA Implementations of Neural Networks - a Survey of a Decade of Progress

Jihan Zhu and Peter Sutton

School of Information Technology and Electrical Engineering,
The University of Queensland, Brisbane, Queensland 4072, Australia

{jihan, p.sutton}@itee.uq.edu.au

Abstract. The first successful FPGA implementation [1] of artificial neural networks (ANNs) was published a little over a decade ago. It is timely to review the progress that has been made in this research area. This brief survey provides a taxonomy for classifying FPGA implementations of ANNs. Different implementation techniques and design issues are discussed. Future research trends are also presented.

1 Introduction

An artificial neural network (ANN) is a parallel and distributed network of simple nonlinear processing units interconnected in a layered arrangement. *Parallelism*, *modularity* and *dynamic adaptation* are three computational characteristics typically associated with ANNs. FPGA-based reconfigurable computing architectures are well suited to implement ANNs as one can exploit concurrency and rapidly reconfigure to adapt the weights and topologies of an ANN.

FPGA realisation of ANNs with a large number of neurons is still a challenging task because ANN algorithms are "multiplication-rich" and it is relatively expensive to implement multipliers on fine-grained FPGAs. By utilizing FPGA reconfigurability, there are strategies to implement ANNs on FPGAs cheaply and efficiently. It is the goal of this paper to: 1) provide a brief survey of existing ANN implementations in FPGA hardware; 2) highlight and discuss issues that are important for such implementations; and 3) provide analysis on how to best exploit FPGA reconfigurability for implementing ANNs. Due to space constraints, this paper can not be comprehensive; only a selected set of papers are referenced.

2 A Taxonomy of FPGA Implementations of ANNs

2.1 Purpose of Reconfiguration

All FPGA implementations of ANNs attempt to exploit the reconfigurability of FPGA hardware in one way or another. Identifying the purpose of reconfiguration sheds light on the motivation behind different implementation approaches.

Prototyping and Simulation exploits the fact that FPGA-based hardware can be rapidly reconfigured an unlimited number of times. This apparent hardware flexibility allows rapid prototyping of different ANN implementation strategies and learning algorithms for initial simulation or proof of concept. The GANGLION project [1] is a good example of rapid prototyping.

Density enhancement refers to methods which increase the amount of effective functionality per unit circuit area through FPGA reconfiguration. This is achieved by exploiting FPGA run-time / partial reconfigurability in one of two ways. Firstly, it is possible to time-multiplex an FPGA chip for each of the sequential steps in an ANN algorithm. For example, in work by Eldredge et al. [2], a back-propagation learning algorithm is divided into a sequence of feedforward presentation and back-propagation stages, and the implementation of each stage is executed on the same FPGA resource. Secondly, it is possible to time-multiplex an FPGA chip for each of the ANN circuits that is specialized with a set of constant operands at different stages during execution. This technique is also known as *dynamic constant folding*. James-Roxby [3] implemented a 4-8-8-4 MLP on a Xilinx Virtex chip by using constant coefficient multipliers with the weight of each synapse as the constant. All constant weights can be changed through dynamic reconfiguration in under $69\mu s$. This implementation is useful for exploiting training-level parallelism with batch-updating of weights at the end of each training epoch. The same idea was also previously explored in work by Zhu et al. [4].

As both methods for density enhancement incur reconfiguration overhead, good performance can only be achieved if the reconfiguration time is small compared to the computation time. There exists a break-even point q beyond which density enhancement is no longer profitable: $q = r/(s-1)$ where r is the time (in cycles) taken to reconfigure the FPGA and s is the total computation time after each reconfiguration [5].

Topology Adaptation refers to the fact that dynamically configurable FPGA devices permit the implementation of ANNs with modifiable topologies. Hence, iterative construction of ANNs [6] can be realized through topology adaptation. During training, the topology and the required computational precision for an ANN can be adjusted according to some learning criteria. de Garis et al. [7] used genetic algorithms to dynamically grow and evolve cellular automata based ANNs. Zhu et al. [8] implemented the Kak algorithm which supports on-line pruning and construction of network models.

2.2 Data Representation

A body of research exists to show that it is possible to train ANNs with ***integer*** weights. The interest in using integer weights stems from the fact that integer multipliers can be implemented more efficiently than floating-point ones. There are also special learning algorithms [9] which use powers-of-two integers as weights. The advantage of powers-of-two integer weight learning algorithms is that the required multiplications in an ANN can be reduced to a series of shift operations. A few attempts have been made to implement ANNs in FPGA hardware with ***floating-point*** weights. However, no successful implementation has been reported to date. Recent work by Nichols et al. [10] showed that despite continuing advances in FPGA technology, it is still impractical to implement ANNs on FPGAs with floating-point precision weights.

Bit-Stream arithmetic is a method which uses a stream of randomly generated bits to represent a real number, that is, the probability of the number of bits that are "on" is the value of the real number. The advantage with this approach is that the required synoptic multiplications can be reduced to simple logic operations. A comprehensive survey of this method can be found in Reyneri [11] while most recent work can found in Hikawa [12]. The disadvantage of bit-stream arithmetic is the lack of precision. This

can severely limit an ANN's ability to learn and solve a problem. In addition, the multiplication between two bit-streams is only correct if the bit-streams are uncorrelated. Producing independent random sources for bit-streams requires large resources.

3 Implementation Issues

3.1 Weight Precision

Selecting weight precision is one of the important choices when implementing ANNs on FPGAs. Weight precision is used to trade-off the capabilities of the realized ANNs against the implementation cost. A higher weight precision means fewer quantization errors in the final implementations, while a lower precision leads to simpler designs, greater speed and reductions in area requirements and power consumption. One way of resolving the trade-off is to determine the "minimum precision" required to solve a given problem. Traditionally, the minimum precision is found through "trial and error" by simulating the solution in software before implementation. Holt and Baker [13] studied the minimum precision required for a class of benchmark classification problems and found that 16-bit fixed-point is the minimum allowable precision without diminishing an ANN's capability to learn these benchmark problems.

Recently, more tangible progress has been made from a theoretical approach to weight precision selection. Draghici [14] relates the "difficulty" of a given classification problem (i.e. how difficult it is to solve) to the required number of weights and the necessary precision of the weights to solve the problem. He proved that, in the worst case, the required weight range $[-p, p]$ is estimated through the minimum distance between patterns of difference classes $d = (\sqrt{n})/(2p)$ to guarantee a solution, where p is an integer and n is the dimension of the input. This services as an important guide for choosing data precision.

3.2 Transfer Function Implementation

Direct implementation for non-linear sigmoid transfer functions is very expensive. There are two practical approaches to approximate sigmoid functions with simple FPGA designs. **Piece-wise linear approximation** describes a combination of lines in the form of $y = ax + b$ which is used to approximate the sigmoid function. Note that if the coefficients for the lines are chosen to be powers of two, the sigmoid functions can be realized by a series of shift and add operations. Many implementations of neuron transfer functions use such piece-wise linear approximations [15]. The second method is **lookup tables**, in which uniform samples taken from the centre of a sigmoid function can be stored in a table for look up. The regions outside the centre of the sigmoid function are still approximated in a piece-wise linear fashion.

4 Conclusion and Future Research Directions

When implementing ANNs on FPGAs one must be clear on the purpose reconfiguration plays and develop strategies to exploit it effectively. The weight and input precision should not be set arbitrarily as the precision required is problem dependent. Future research areas will include: **benchmarks**, which should be created to compare and ana-

lyse the performance of different implementations; **software tools**, which are needed to facilitate the exchange of IP blocks and libraries of FPGA ANN implementations; **FPGA friendly learning algorithms**, which will continue to be developed as faithful realizations of ANN learning algorithms are still too complex and expensive; and, **topology adaptation** approaches, which take advantage of the features of the latest FPGAs such as specialized multipliers and MAC units.

References

1. Cox, C.E. and E. Blanz, *GangLion - a fast field-programmable gate array implementation of a connectionist classifier.* IEEE Journal of Solid-State Circuits, 1992. **28**(3): p. 288-299.
2. Eldredge, J.G. and B.L. Hutchings. *Density enhancement of a neural network using FPGAs and run-time reconfiguration*. in *Proceedings of IEEE Workshop on Field-Programmable Custom Computing Machines,* 1994. pp 180-188.
3. James-Roxby, P. and B.A. Blodget. *Adapting constant multipliers in a neural network implementation*. in *Proceedings of IEEE Symposium on Field-Programmable Custom Computing Machines,* 2000. pp 335-336.
4. Zhu, J.M., G.J.; Gunther, B.K., *Towards an FPGA based reconfigurable computing environment for neural network implementations*, in *Proceedings of Ninth International Conference on Artificial Neural Networks,* 1999. pp. 661-666, vol.2.
5. Guccione, S.A. and M. Gonzalez. *Classification and performance of reconfigurable architectures*. in *Proceedings of the 5th International Workshop on Field-Programmable Logic and Applications*. 1995, pp 439-448, Springer-Verlag, Berlin.
6. Perez-Uribe, A. and E. Sanchez. *FPGA Implementation of an Adaptable-Size Neural Network*. in *Proceedings of the Sixth International Conference on Artificial Neural Networks*. 1996. pp 382-388, Springer-Verlog.
7. de Garis, H., et al. *Initial evolvability experiments on the CAM-brain machines (CBMs)*. in *Proceedings of the 2001 Congress on Evolutionary Computation,* 2001. pp 635-641, vol. 1.
8. Zhu, J. and G. Milne. *Implementing Kak Neural Networks on a Reconfigurable Computing Platform*. in *Proceedings of the 10th International Workshop on Field-Programmable Logic and Applications - Roadmap to Reconfigurable Computing*. 2000. pp 260- 269. Springer Verlag.
9. Marchesi, M., et al., *Fast neural networks without multipliers.* IEEE Transactions on Neural Networks, 1993. **4**(1): p. 53-62.
10. Nichols, K., M. Moussa, and S. Areibi. *Feasibility of Floating-Point Arithmetic in FPGA based Artificial Neural Networks*. in *Proceedings of the 15th International Conference on Computer Applications in Industry and Engineering*. 2002. pp San Diego, California.
11. Reyneri, L.M. *Theoretical and implementation aspects of pulse streams: an overview*. in *Proceedings of the Seventh International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*. 1999. pp 78-89.
12. Hikawa, H., *A new digital pulse-mode neuron with adjustable activation function.* IEEE Transactions on Neural Networks, 2003. **14**(1045-9227): p. 236-242.
13. Holt, J.L., T.E. Baker. *Back propagation simulations using limited precision calculations,* in *Proceedings of International Joint Conference on Neural Networks*. 1991. pp 121-126 vol. 2.
14. Draghici, S., *On the capabilities of neural networks using limited precision weights.* Neural Networks, 2002. **15**: p. 395-414.
15. Wolf, D.F., Romero, R. A. F., Marques, E. *Using Embedded Processors in Hardware Models of Artificial Neural Networks*. in *In proceedings of SBAI - Simpósio Brasileiro de Automao Inteligente*. 2001. pp 78-83.